OO – Engenharia Eletrônica

-

Programação em C/C++

Slides 20: TCP/IP em Winsocks 2.

API do Windows para programar aplicativos
que utilizam o protocolo TCP/IP .

Prof. Jean Marcelo SIMÃO

TCP/IP em Winsocks 2

Usando a API do Windows para programar aplicativos que utilizam o protocolo TCP/IP

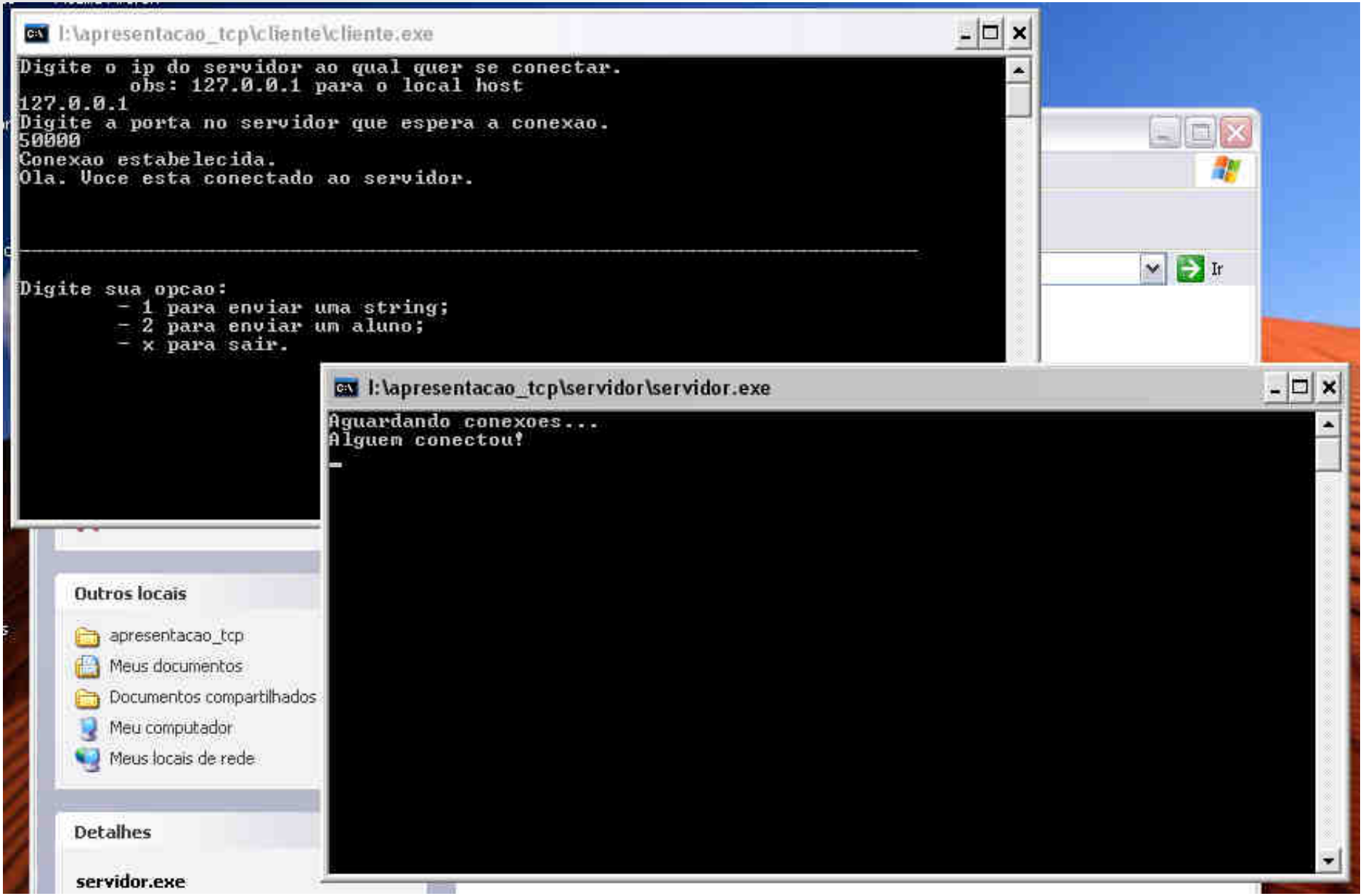
Obs.: Material inicial elaborado por André de Castilho Costa Pinto no 2o Semestre de 2007. O Castilho era então aluno da disciplina em questão e estagiário em projeto do Professor da disciplina.

Programa exemplo

Explicações prévias:

- Toda comunicação depende de dois participantes: quem envia dados e quem recebe dados;
- O "programa exemplo", portanto, é na verdade dois programas. Um *servidor* e um *cliente.*
- Neste caso o servidor é quem recebe certas informações e o cliente é quem envia tais informações. O cliente se conecta ao servidor para tal.

Programa exemplo



There's no place like 127.0.0.1

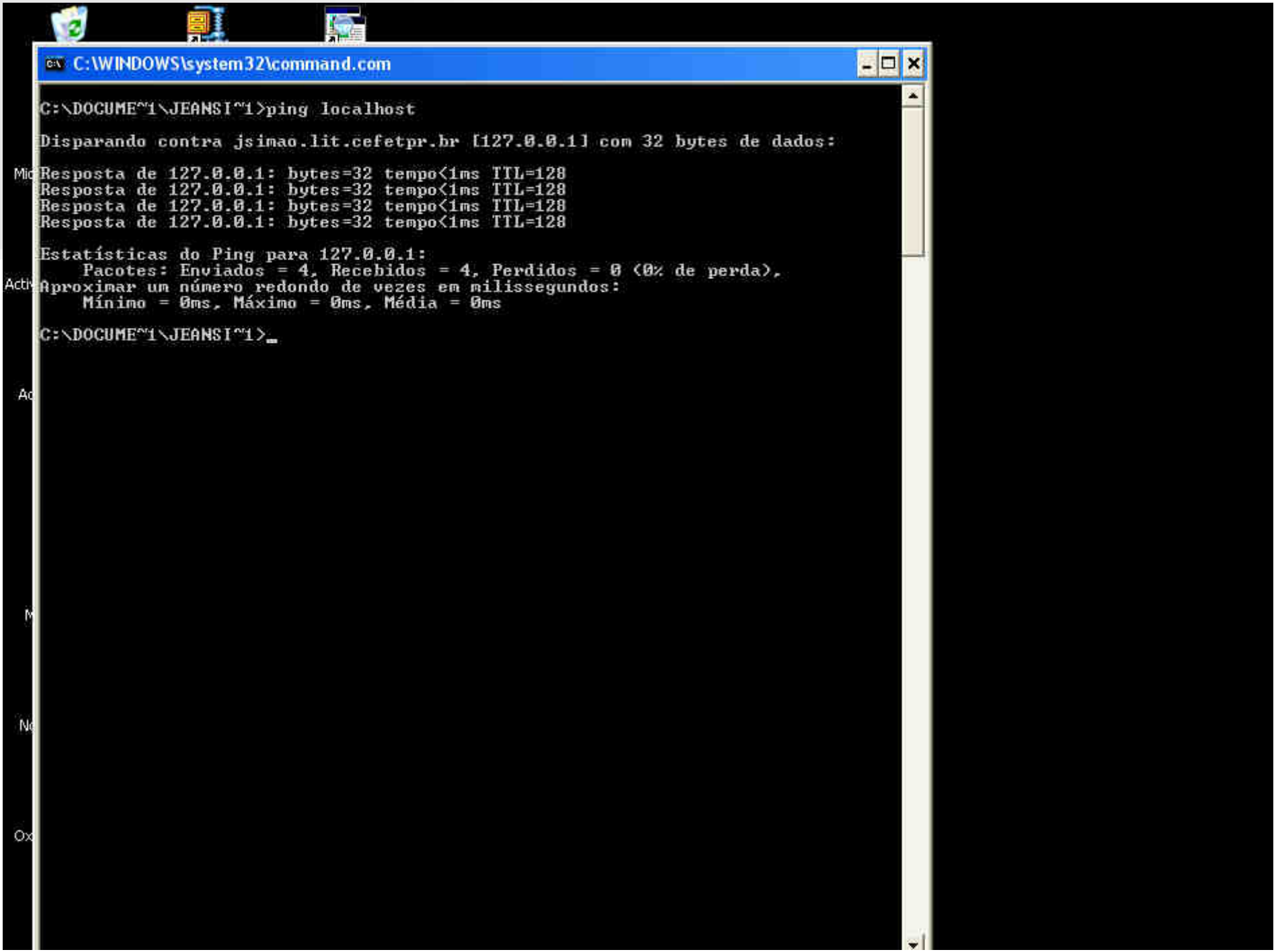
- Vocês devem ter percebido que os dois programas estão rodando no mesmo computador... Acontece que, para não precisar montar uma rede toda vez que formos testar programas TCP/IP, podemos usar o endereço: 127.0.0.1.

- Este endereço de *IP* é chamado de *localhost* ou *loopback,* e referencia o próprio computador.

There's no place like 127.0.0.1

- Uma prova prática disso é a seguinte:

1. Vá em Iniciar → Executar → cmd.
2. Desconecte o cabo da rede.
3. Digite "*ping localhost*", "*ping 127.0.0.1*" ou ainda "*ping loopback*".
4. Veja que o *PC*, mesmo sem estar conectado a nada, é capaz de pingar esse *IP*.


C:\WINDOWS\system32\command.com
C:\DOCUME~1\JEANSI~1>ping localhost
Disparando contra jsimao.lit.cefetpr.br [127.0.0.1] com 32 bytes de dados:
Resposta de 127.0.0.1: bytes=32 tempo<1ms TTL=128
Resposta de 127.0.0.1: bytes=32 tempo<1ms TTL=128
Resposta de 127.0.0.1: bytes=32 tempo<1ms TTL=128
Resposta de 127.0.0.1: bytes=32 tempo<1ms TTL=128
Estatísticas do Ping para 127.0.0.1:
Pacotes: Enviados = 4, Recebidos = 4, Perdidos = 0 (0% de perda),
Aproximar um número redondo de vezes em milissegundos:
Mínimo = 0ms, Máximo = 0ms, Média = 0ms
C:\DOCUME~1\JEANSI~1>

Antes de programar...

- Devemos conhecer:

1. Alguma ferramenta que monitora as portas;

2. Algum modo de descobrir o endereço de *IP* (*Internet Protocol*) de um computador.

Usando o bom e velho DOS

- O *prompt* de comando do *Windows* manteve vários comandos úteis do *DOS*, e, para monitorar o estado das portas do computador, utilizaremos o *netstat*.

- Este comando exibe uma lista das conexões estabelecidas no computador.

Usando o bom e velho DOS

- Para descobrir qual o *ip* de uma máquina, a maneira mais rápida é ir no *prompt* de comando e digitar *ipconfig*;

Antes de programar...

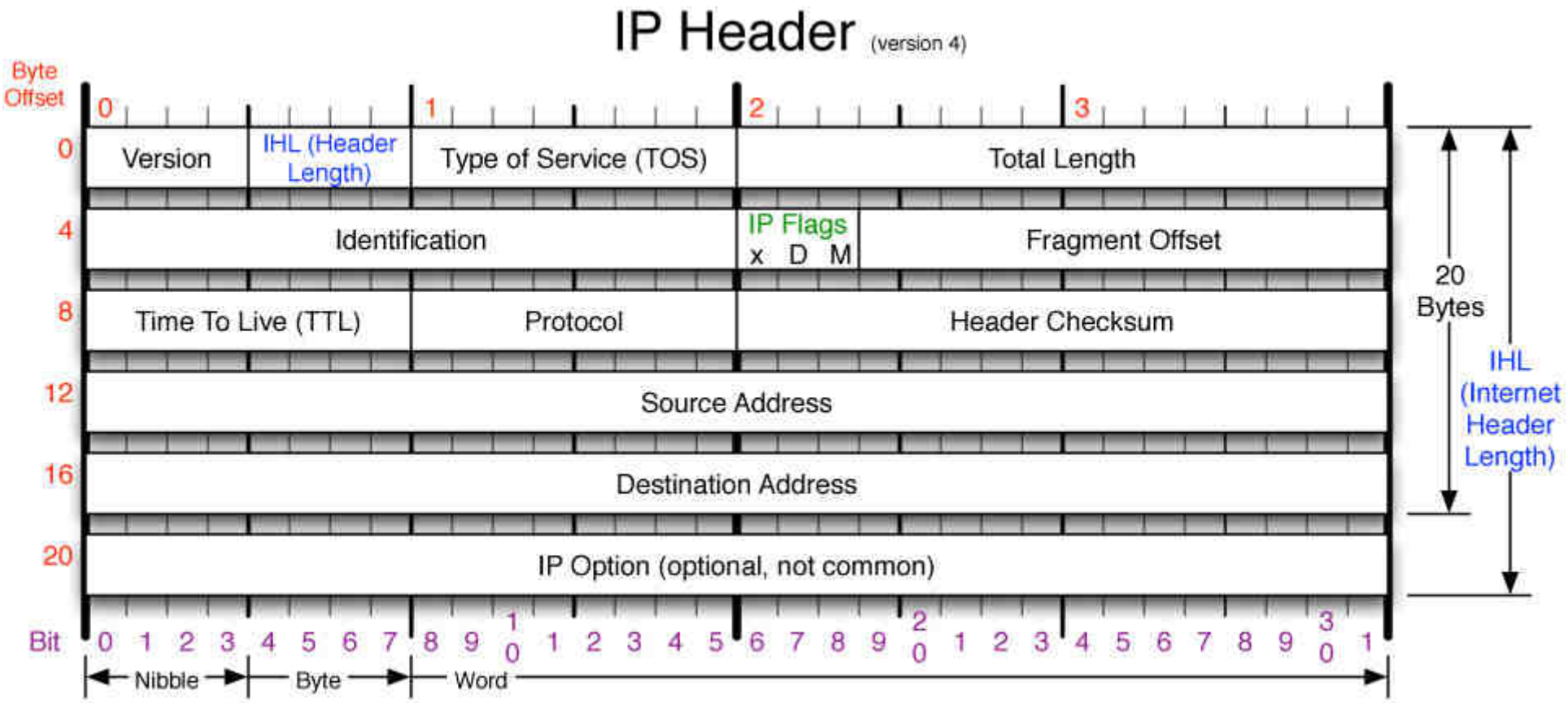
- Devemos saber o que é:

1. IP (*Internet Protocol*);
2. TCP (*Transfer Control Protocol*);
3. Socket;
4. Porta.

Internet Protocol Suite

- Protocolo de comunicação utilizado pela internet para estabelecer a conexão remota entre computadores. O protocolo de rede mais utilizado pela internet nos dias de hoje é o IPv4, mas poderá ser substituído pelo IPv6.

O protocolo IP



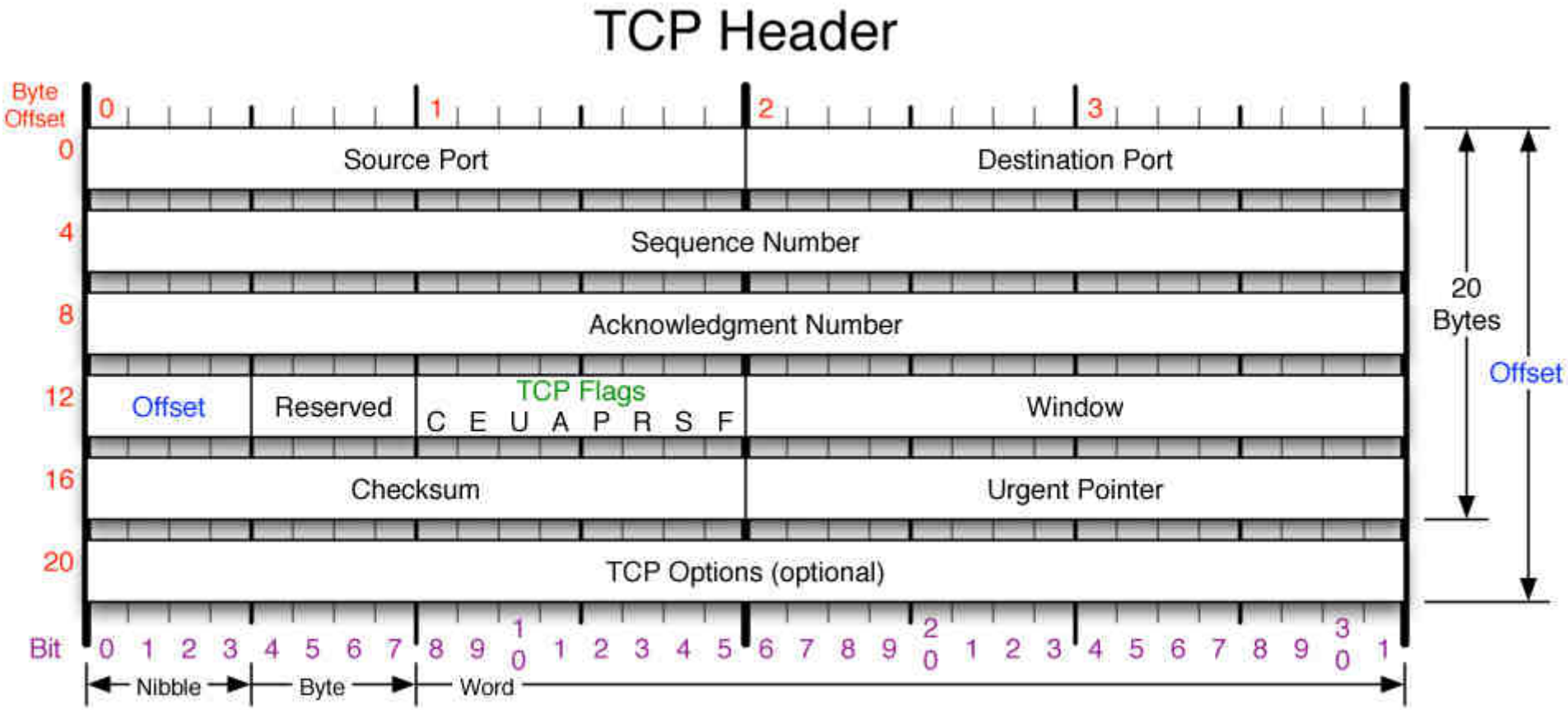
Protocolos de Transporte

- Os mais difundidos são o *TCP* e o *UDP*.

- *TCP – Transmission Control Protocol*
  - Ele é usado por outros protocolos de alto nível, como o *FTP* e o *HTTP*)

- *UDP – User Datagram Protocol*

O protocolo TCP



Porta

- Número que associa os pacotes recebidos a um determinado processo rodando na máquina.

Socket

- É a união das duas informações necessárias à comunicação nos protocolos UDP e TCP.

- Um *socket* é formado por um endereço IP e uma porta.

E como se programa?

- Programa-se via *API* do *Windows*.
  - A *API* (*Application Programming Interface*) do *Windows* constitui-se em uma bibliotecas de funções que permite a qualquer um escrever códigos que rodem em *Windows.*

- A *API* inclui uma biblioteca que contém as funções que gerenciam e utilizam *sockets* para estabelecer conexões ou enviar pacotes de dados.

Primeiro Passo

- Já que vamos usar essa biblioteca, devemos dizer isso ao compilador. Deste modo, ele poderá reconhecer as funções que utilizaremos no programa.

- A biblioteca em questão é a "ws2_32.lib", abreviação de winsocks2, para plataformas win32.

- Essas funções são implementadas em uma *DLL* (*Dynamic Link Library*) que já vem com o Windows.

  - *Dynamic Link Library* ou Biblioteca de Ligação Dinâmica (*DLL*) é uma biblioteca de ligação ["*linkagem*"] dinâmica contendo códigos ou dados, que podem ser compartilhados por diferentes programas aplicativos durante sua execução.

    http://www.arnaut.eti.br/op/CPPGlos.htm, visitado em 29/05/2008, às 9:22.

Enfim, como fazer

- As funções básicas são:

1. socket (...)
2. bind (...)
3. listen (...)
4. accept (...)
5. connect (...)
6. send (...)
7. recv (...)
8. WSAStartup (...)
9. closesocket (...)

O código

O código – Versão 1

O código – Versão 1

A parte do Servidor

Algumas classes "já conhecidas' que estão no projeto do servidor

```
#ifndef _ALUNO_H_
#define _ALUNO_H_
#include "Pessoa.h"
class Aluno : public Pessoa
{
private:
   int          id;
   int          RA;
public:
   Aluno (      int diaNa, int mesNa, int anoNa,
                char* nome = "", int i = 0, int r = 0 );
   Aluno (  int i  );
   Aluno ( );
   ~Aluno ( );
   void setRA ( int ra );
   int getRA ();
   void setId (int I );
   int getId ( );
};
#endif
```

```
#ifndef _PROFESSOR_H_
#define _PROFESSOR_H_
#include "Pessoa.h"
class Professor : public Pessoa
{
private:
   float          salario;
   float          bolsa_projeto;
public:
   Professor (   int diaNa, int mesNa, int anoNa,
                 char* nome = "", float sal = 0,  float bp = 0);
    Professor ( );
    ~Professor ( );
    void setSalario ( float s );
    float getSalario ( );
    void setBolsaProjeto ( float bp );
    float getBolsaProjeto ( );
    void informaProventos ( );
};
#endif
```

Outras classes "já conhecidas' que estão no projeto do servidor

```
#ifndef _PESSOA_H_
#define _PESSOA_H_

class Pessoa
{
 private:

   . . .

public:

   . . .

};
#endif
```

```
#ifndef _LISTA_H_
#define _LISTA_H_
#include "Elemento.h"
#include <iostream> using namespace std;
template<class TIPO>
class Lista
{private:  . . .
public:   . . .
};
. . .
```

```
#ifndef _ELEMENTO_H_
#define _ELEMENTO_H_
template<class TIPO>
class Elemento
{
private:  . . .
public:  . . .
};
. . .
```

Classe Servidor – em *servidor.h*

```
#ifndef _SERVIDOR_H_
#define _SERVIDOR_H_

#include <winsock2.h>
#include <windows.h>
#include <process.h>
#include "Lista.h"
#include "Aluno.h"
#include "Professor.h"

class Servidor
{
   sockaddr_in meu_end;
   long int cont_msg;
   SOCKET meu_Sockt;
   bool tem_Canal,  tem_Sockt,  tem_Conexao;
   int max_conexao,  bytes_recebidos;  // bytes recebidos, armazena o valor de retorno de recv
   char buffer [ 500 ];
   Lista < Aluno > lAlunos;
   Lista < Professor > lProfessor;

public:
    Servidor ( char* ip = "127.0.0.1", unsigned short porta = 3077, int max = 1);
   ~Servidor ( );

   void conectaSockt ();
   /*-------------------------------------------------------------------------*/
   void aceitaSockt ( ) ;
   void lacoDeConexao ( );
   /*-------------------------------------------------------------------------*/
   void recebeClasse ( );
   void cadastraAluno ( );
   void cadastraProfessor ( );
};
#endif
```

Classe Servidor – *construtora* – em *servidor.cpp*

```
#include "servidor.h"
Servidor::Servidor ( char *ip , unsigned short porta , int max ) : cont_msg ( 0 )
{   WSADATA tst;
   if ( WSAStartup ( MAKEWORD ( 2 , 2 ) , &tst ) )  // teste para ver se o computador suporta e versão de winsocks utilizada
   {  cout << " O computador nao possui a versao 2.0 do Winsocks. " ;
      cout << " Nao sera possivel criar o servidor. " << endl;
   }
   else
   {   max_conexao = max;
      // Atribui os valores passados pelo construtor ao ip e ao endereço de porta
      meu_end.sin_family          =     AF_INET;
      meu_end.sin_addr.s_addr   =     inet_addr ( ip );      // AdressFamily-Internet será usado como padrão nesse programa.
      meu_end.sin_port            =     htons ( porta );
      meu_Sockt = socket ( AF_INET, SOCK_STREAM, IPPROTO_TCP );  // cria o socket;
      if ( meu_Sockt == SOCKET_ERROR )  {
         cout << " Erro na criacao do socket." << endl;
         tem_Sockt = false;
      }
      else  {
          tem_Sockt = true;
      }
   }
}
```

Classe Servidor – *destrutora* e *conectaSockt* – em *servidor.cpp*

```
Servidor::~Servidor ()
{
   closesocket ( meu_Sockt );
}

void Servidor::conectaSockt ()
{
   int result = -1;
   result = bind  (
                        meu_Sockt ,
                        reinterpret_cast < SOCKADDR* > ( &meu_end ) ,
                        sizeof ( meu_end )
                  ) ;
   if ( result == -1 )
   {
      cout << " Bind nao pode ser efetuado." << endl ;
      tem_Canal = false ;
   }
   else
   {
      tem_Canal = true ;
   }
}
```

Classe Servidor – *aceitaSockt* – em *servidor.cpp*

```
void Servidor::aceitaSockt ()
{
   SOCKET outroSockt;

   // listen define o estado da porta aberta pelo servidor.
   // a porta fica esperando ("escutando") pedidos de conexões

   if ( listen ( meu_Sockt , max_conexao ) == -1 )
   {
      cout << "Erro ao entrar em modo de espera de conexoes" << endl;
      tem_Conexao = false;
      return;
   }

   cout << "Aguardando conexoes..." << endl;

   // loop infinito...
   do
   {

      // função que aceita um pedido de conexão feito com "connect()" pelo cliente
      outroSockt = accept( meu_Sockt, NULL, NULL );

   } while ( outroSockt == SOCKET_ERROR );

   meu_Sockt = outroSockt;

   cout << "Alguem conectou!" << endl;

   // envia uma mensagem confirmando a conexão para o cliente
   send ( meu_Sockt, "Ola. Voce esta conectado ao servidor.", 38, 0 );

}
```

Classe Servidor – *lacoDeConexao* – em *servidor.cpp*

```
//função que define o loop que ficará esperando por mensagens do cliente

void Servidor::lacoDeConexao()
{
   while (1)
   {
      bytes_recebidos = recv ( meu_Sockt, buffer, 500, 0 );

      switch ( bytes_recebidos )
      {
         case -1: continue;

         case 2 :  {  recebeClasse ();
                      cout << "----------------------------------------------------------------------" << endl;
                   }  break;

         case 0:   {  cout << " Mensagem num. " << ++cont_msg << endl;
                      cout << "\t Conexão fechada pelo cliente." << endl;
                      shutdown ( meu_Sockt, 1 );
                      closesocket( meu_Sockt );
                      system ( "pause" );
                      WSACleanup ();
                      exit ( EXIT_SUCCESS );
                   }

         default: {  cout << "Mensagem num. " << ++cont_msg << endl;
                      cout << "\t String de texto recebida.";
                      cout << "\tA mensagem foi: \";
                      cout << buffer << "\"" << endl;
                   }
      }
      strcpy ( buffer, "" );
   }
}
```

Classe Servidor – *recebeClasse* – em *servidor.cpp*

```
void Servidor::recebeClasse ()
{
   char *resp;
   resp = new char [ 2 ];
   strcpy ( resp, "1" );
   send ( meu_Sockt, resp, 2, 0 );

   int tipo = atoi ( buffer );

   do
   {
       bytes_recebidos = recv ( meu_Sockt, buffer, 500, 0 );
   }
   while ( bytes_recebidos == -1 );

   switch ( tipo )
   {
      case 1:   { cadastraAluno();         } break;
      case 2:   { cadastraProfessor();     } break;
      default:  { cout << "Houve uma falha durante o recebimento, os dados podem estar corrompidos." << endl; } break;
   }
   lacoDeConexao ();
}
```

Classe Servidor – cadastraAluno - cadastraProfessor

```
void Servidor::cadastraAluno()
{
    Aluno *pAux;

    pAux = reinterpret_cast < Aluno  * > ( buffer );

    lAlunos.setInfo ( pAux, pAux->getNome () );

    cout << "Msg num." << ++cont_msg << endl;

    cout << " \tAluno recebido e listado!" << endl;

    cout << " \tO nome do aluno eh " << pAux->getNome() << endl;

}

void Servidor::cadastraProfessor ()
{

    Professor *pAux;

    pAux = reinterpret_cast < Professor *>( buffer);

    lProfessor.setInfo ( pAux, pAux->getNome() );

    cout << "Msg num." << ++cont_msg << endl;

    cout << " \tProfessor recebido e listado!" << endl;

    cout << " \tO nome do professor eh " << pAux->getNome() << endl;

}
```

main ()

```
#include <iostream>
#include <string>
#include <stdlib.h>

#include "servidor.h"
using std::cout;
using std::endl;

main()
{

   Servidor serv;
   serv.conectaSockt ( );

   serv.aceitaSockt ();
   serv.lacoDeConexao ();

   system ( "pause" );
}
```

O código – Versão 1

A parte do Cliente

Algumas classes "já conhecidas' que estão no projeto do cliente

```
#ifndef _PESSOA_H_
#define _PESSOA_H_
...
class Pessoa
{
 private:
   ...
public:
   ...
};
#endif
```

```
#ifndef _ALUNO_H_
#define _ALUNO_H_
#include "Pessoa.h"
class Aluno : public Pessoa
{
private:
   ...
public:
   ...
};
#endif
```

```
#ifndef _PROFESSOR_H_
#define _PROFESSOR_H_
#include "Pessoa.h"
class Professor : public Pessoa
{
private:
   ...
public:
   ...
};
#endif
```

A classe *Cliente* - em *cliente.h*

```
#ifndef _CLIENTE_H_
#define _CLIENTE_H_
#include <winsock2.h>
#include <windows.h>
#include "Aluno.h"
#include "Pessoa.h"
#include "Professor.h"
class Cliente
{
   sockaddr_in       serv_end;
   sockaddr_in       meu_end;              // OBS: na verdade, nao precisa ser definido o endereço do cliente
   SOCKET            meu_sockt;
public:
   Cliente ();
   ~Cliente ();
   void conectar  (  char *ip, unsigned short porta  );       // A função que não recebe parâmetros pergunta ao usuario a porta e o ip do server
   void conectar  (  );
   void enviaString   ( );                                    // Envia uma string de texto
   void criaEnviaAluno  ( char *nome, int RA, int ID );       // Envia um buffer contendo os dados referentes a um aluno;
   void criaEnviaProfessor ( char *nome, float sal, float bp );
   void rotinaPrincipal ( );                                  // Rotina principal de envio de dados
};
#endif
```

A construtora da classe *Cliente* – em *cliente.cpp*.

```
#include <iostream>
using namespace std;
#include "cliente.h"
Cliente::Cliente ( )
{
   WSADATA tst;
   if  ( WSAStartup  (  MAKEWORD ( 2 , 2 ) , &tst ) )  // teste para ver se o computador suporta e versão de winsocks utilizada
   {                                                   // não deve dar problemas, já que todos os windows superiores ao 95 a suportam.
      cout << "O computador nao possui a versao 2.0 do Winsocks.";
      cout << " Nao sera possivel criar o servidor." << endl;
      return;
   }
   meu_sockt = socket ( AF_INET,  SOCK_STREAM, IPPROTO_TCP );          // cria o socket.
   if  ( meu_sockt == SOCKET_ERROR )
   {
     cout << "Nao foi possivel criar o socket." << endl;
   }
   // define o endereço do cliente.
   // existem funções que recuperam os ips reais de uma máquina nas respectivas redes das quais participa
   meu_end.sin_family                  = AF_INET;
   meu_end.sin_port                    = htons ( 50001 );
   meu_end.sin_addr.s_addr             = inet_addr ( "127.0.0.1" );
}
```

A destrutora da classe *Cliente* – em *cliente.cpp*.

```
Cliente::~Cliente ( )
{
   closesocket ( meu_sockt );
}
```

O método *conectar ( )* da classe *Cliente* – em *cliente.cpp*.

```
void Cliente::conectar ( )
{
   char msg [ 38 ];
   char ip [ 16 ];

   cout << " Digite o ip do servidor ao qual quer se conectar. " << endl;
   cout << " \t obs: 127.0.0.1 para o local host \n";

   fflush ( stdin );

   gets ( ip );

   conectar ( ip, 3077 );

}
```

O método *conectar (. . .)* da classe *Cliente* – em *cliente.cpp*.

```
void Cliente::conectar ( char* ip, unsigned short porta )
{
   char msg [38] ;
   serv_end.sin_family = AF_INET;                    // define o endereço do servidor.
   serv_end.sin_addr.s_addr = inet_addr ( ip );
   serv_end.sin_port = htons ( porta );

   int result = 0;
   result = connect ( meu_sockt, reinterpret_cast < SOCKADDR * > ( &serv_end ), sizeof ( serv_end ) );
   if ( result == -1 )
   {
      cout << " Nao foi possivel conectar ao servidor, tente de novo." << endl;
      return;
   }
   cout << " Conexao estabelecida." << endl;

   while ( 1 )
   {
      result = recv ( meu_sockt, msg, 38, 0 );
      if ( result != -1 )
      {
         cout << msg << endl << endl;
         break;
      }
   }
   rotinaPrincipal ( );
}
```

O método *rotinaPrincipal ( )* da classe *Cliente* – em *cliente.cpp*.

```
// rotina de envio, que contém o menu e a interface do usuario
void Cliente::rotinaPrincipal ( )
{  char opc, nom [ 100 ];
   int ra, id;
   float sal, bp;

   cout << "Digite sua opcao:"                 << endl;
   cout << "\t- 1 para enviar uma string;"     << endl;
   cout << "\t- 2 para enviar um aluno;"       << endl;
   cout << "\t- 3 para enviar um professor;"   << endl;
   cout << "\t- x para sair."                  << endl;
   cin >> opc;

   switch ( opc )
   {
     case '1':  {
         enviaString ();
         system ( "cls" );
         rotinaPrincipal ();
     } break;

     case '2': {
       cout      << "Digite o nome do aluno que deseja cadastrar."
                 << endl;
       flush ( stdin );
       gets ( nom );
       cout      << "Digite o RA." << endl;
       cin       >> ra;
       cout      << "Digite a ID." << endl;
       cin       >> id;
       criaEnviaAluno ( nom, ra, id );
       system ( "cls" );
       rotinaPrincipal ( );
     }  break;
```

```
     case '3':
     {    cout << "Digite o nome do prof. que deseja cadastrar."
               << endl;

          fflush ( stdin );

          gets ( nom );

          cout << " Digite o salario." << endl;
          cin >> sal;

          cout << "Digite o valor da bolsa projeto." << endl;
          cin >> bp;

          criaEnviaProfessor ( nom, sal, bp );
          system ( "cls" );

          rotinaPrincipal ();
     } break;

     case 'x':
     case 'X':  {   shutdown ( meu_sockt, 1 );
                    closesocket ( meu_sockt );
                    system ("Pause");
                    exit ( EXIT_SUCCESS );
                }  break;

     default: {    cout << "Opcao invalida." << endl;
                    system ( "cls" );
                    rotinaPrincipal ( );
                }   break;
   }
}
```

O método *enviaString( )* da classe *Cliente* – em *cliente.cpp*.

```
void Cliente::enviaString ( )
{
   char msg [ 500 ];
   cout << " Digite a msg que deseja enviar: " << endl;
   fflush ( stdin );
   gets ( msg );

   send (
                  meu_sockt ,
                  msg ,
                  strlen ( msg ) + 1 ,
                  0
        );
}
```

O método *cadastraAluno (...)* da classe *Cliente* – em *cliente.cpp*.

```
void Cliente::criaEnviaAluno ( char* nome, int RA, int ID )
{
   Aluno *aux;              // é criado o objeto da classe aluno que será enviado;
   char *buff;              // um buffer que conterá os dados do aluno;
   int    tipo;
   char *inf;

   inf = new char [ 2 ];

   strcpy ( inf ,  "1");                    //  Rotina que diz ao servidor qual o tipo de dado que está sendo enviado

   send ( meu_sockt , inf , 2 , 0 );        //  inf recebe 1 para sinalizar que é um aluno, e 2 que é um professor

   while ( recv ( meu_sockt , inf , 2 , 0 ) == -1 );   //  fica em loop até receber resposta do servidor

   tipo = atoi ( inf );

   if  ( tipo == 1 )
   {
      aux =      new Aluno ( 0 , 0 , 0 , nome , RA , ID );
      // o buffer é inicializado com o tamanho (em bytes) da estrutura aluno
      // como 1 char tem 1 byte, todo o vetor "buff" terá o mesmo tamanho em bytes que um objeto aluno;
      buff =     new char [ sizeof ( Aluno ) ];

      // dizemos ao compilador que, em vez dele entender aux como um aluno, ele deve interpretá-lo como um vetor de char.
      // Ou seja, estamos interpretando aux não como um objeto aluno, mas como um vetor de bytes.
      buff =     reinterpret_cast < char * > ( aux );

      // agora podemos enviar os bytes, que, juntos, representam um aluno, para o servidor.
      send       ( meu_sockt , buff , sizeof ( Aluno ) , 0 );
      system     ( "pause" );
   }
   else
   {
      cout << "Nao foi possivel enviar o aluno." << endl;
   }
}
```

O método *cadastraProfessor (...)* da classe *Cliente* – em *cliente.cpp*.

```
void Cliente::criaEnviaProfessor ( char *nome, float sal, float bp )
{
    Professor *aux;
    char      *buff;
    char      *inf;
    int       tipo;

    inf = new char [ 2 ];

    strcpy ( inf , "2" );

    send ( meu_sockt , inf ,  2 , 0 );

    while ( recv ( meu_sockt , inf , 2 , 0 ) == -1 );

    tipo = atoi ( inf );

    if ( tipo == 1 )
    {
       aux = new Professor ( 0 , 0 , 0 , nome, sal, bp );

       buff = new char[ sizeof ( Professor ) ];
       buff = reinterpret_cast < char * > ( aux );

       send ( meu_sockt , buff , sizeof ( Professor ) , 0 );

       delete aux;
    }
    else
    {
       cout << "Nao foi possivel enviar o professor." << endl;
    }
}
```

A função *main()* – em *main.cpp*.

```
#include "cliente.h"
#include <iostream>

#include <stdlib.h>
using std::cout;
using std::endl;

int main ( )
{

   Cliente cl;

   cl.conectar ();

   system ( "pause" ) ;

   return 0;

}
```

O código – Versão 2

usando Threads.

O código – Versão 2
A parte do Servidor

A parte do servidor mudou ligeiramente no *servidor.h* e *.cpp* e mudou na *main.cpp*.

Mudanças no servidor *.h* e *.cpp*.

```
. . .
class Servidor
{
   CRITICAL_SECTION *cs;
   int *iter;
   . . .

public:
   Servidor  (
                  CRITICAL_SECTION *pcs,
                  int *it,
                  char *ip = "127.0.0.1",
                  unsigned short porta = 3077,
                  int max = 1
               );
  . . .

   void listaAlunos ( ) ;
   void listaProfessores ( ) ;

};
. . .
```

```
. . .

void Servidor::listaAlunos ( )
{

    lAlunos.listeInfos ( );

}

void Servidor::listaProfessores ( )
{

    lProfessor.listeInfos ( );

}
```

```
Servidor::Servidor ( CRITICAL_SECTION *pcs, int *it, char *ip, unsigned short porta, int max ): cont_msg ( 0 )
{
   WSADATA tst;
   cs = pcs;
   iter = it;

   if  ( WSAStartup ( MAKEWORD ( 2 , 2 ), &tst ) ) // teste para ver se o computador suporta e versão de winsocks utilizada
   {
      cout << " O computador nao possui a versao 2.0 do Winsocks. ";
      cout << " Nao sera possivel criar o servidor. " << endl;
   }
   else
   {
      max_conexao = max;

      // Atribui os valores passados pelo construtor ao ip e ao endereço de porta
      // AdressFamily-Internet será usado como padrão nesse programa
      meu_end.sin_family = AF_INET;
      meu_end.sin_addr.s_addr = inet_addr ( ip );
      meu_end.sin_port = htons ( porta );

      // cria o socket;
      meu_Sockt = socket ( AF_INET, SOCK_STREAM, IPPROTO_TCP );

      if  ( meu_Sockt == SOCKET_ERROR )
      {
         cout << " Erro na criacao do socket." << endl;
         tem_Sockt = false;
      }
      else
      {
         tem_Sockt = true;
      }
   }
}
```

```
void Servidor::aceitaSockt ( )
{
   SOCKET outroSockt;
   // listen define o estado da porta aberta pelo servidor.
   // a porta fica esperando ("escutando") pedidos de conexões

   if ( listen ( meu_Sockt, max_conexao ) == -1 )
   {
      cout << "Erro ao entrar em modo de espera de conexoes" << endl;
      tem_Conexao = false;
      return;
   }

   EnterCriticalSection ( cs );
   cout << "Aguardando conexoes...\n" << endl;
   LeaveCriticalSection ( cs );
   Sleep ( 10 );

   // loop infinito... dá para implementar, por meio de comandos que façam o thread parar
   // de ser executado e uma variável de controle, um método que gerencie um possível "time out“
   do
   {
      // função que aceita um pedido de conexão feito com "connect()" pelo cliente
      outroSockt = accept ( meu_Sockt, NULL, NULL );
   } while ( outroSockt == SOCKET_ERROR );

   meu_Sockt = outroSockt;

   EnterCriticalSection ( cs );
   cout << " Alguem conectou! " << endl;
   LeaveCriticalSection ( cs );
   Sleep ( 10 );

   // envia uma mensagem confirmando a conexão para o cliente
   send ( meu_Sockt, “ Ola. Voce esta conectado ao servidor. ", 38, 0 );
}
```

```
// função que define o loop que ficará esperando por mensagens do cliente
void Servidor::lacoDeConexao ( )
{
   while ( *iter )
   {  bytes_recebidos = recv ( meu_Sockt, buffer, 500, 0 );
      if  ( bytes_recebidos != -1 )
      {
         EnterCriticalSection ( cs );
         switch ( bytes_recebidos )
         {  case  2:
            {
               recebeClasse ( );
               cout << "----------------------------------------------------------------" << endl;
            }   break;
            case 0:
            {   cout << " Mensagem num. " << ++cont_msg << endl;
                cout << "\t Conexão fechada pelo cliente."    << endl;
               shutdown ( meu_Sockt, 1 );
               closesocket ( meu_Sockt );
               system ( "pause" );
               WSACleanup ( );
               exit ( EXIT_SUCCESS );
            }   break;
            default:
            {   cout << "Mensagem num. " << ++cont_msg << endl;
                cout << "\t String de texto recebida.";
                cout << " \tA mensagem foi: \ ";
                cout << buffer << "\"" << endl;  }
         }
         LeaveCriticalSection ( cs );
         Sleep ( 10 );
      }
      strcpy ( buffer, "" );
   }
   cout << " Conexao fechada." << endl;
}
```

Mudanças no *main.cpp*

```
#include <windows.h>

CRITICAL_SECTION cs;

int iter = 1;
// flag que sinaliza o processo de loopConexao
// caso o usuario queira sair do programa

#include "servidor.h"
void interfaceUsuario ( void *param ) ;

main ( )
{
   Servidor serv ( &cs , &iter );
   InitializeCriticalSection ( &cs );

    serv.conectaSockt ( );
   _beginthread ( interfaceUsuario, 0, &serv );

   serv.aceitaSockt ( );
   serv.lacoDeConexao ( );

   WSACleanup ( ); // fecha portas, socks...
   system ( "pause" );
}
```

```
void interfaceUsuario ( void *param )
{
   int opc;
   Servidor *serv1;
   serv1 = reinterpret_cast < Servidor *> ( param );

   while ( iter )
  {
     EnterCriticalSection ( &cs );
     cout <<    " Servidor versao 0.2: "        << endl;
     cout <<    " \t    Qual sua opcao? "       << endl;
     cout <<    " \t    1 - Listar Alunos"      << endl;
     cout <<    " \t    2 - Listar Professores"; << endl;
     cout <<    " \n\t  3 - Sair" << endl;
     LeaveCriticalSection ( &cs );
     cin >> opc;
     switch ( opc )
     {
        case 1: {  EnterCriticalSection ( &cs );
                   serv1->listaAlunos ( );
                   cout << endl;    system ( "pause" );
                   LeaveCriticalSection ( &cs );
                } break;
       case 2: {   EnterCriticalSection ( &cs );
                   serv1->listaProfessores ();
                   cout << endl;      system ( "pause" );
                   LeaveCriticalSection ( &cs );
                } break;
       case 3: {  WSACleanup ();
                   exit ( EXIT_SUCCESS );
                }   break;
       default: {  system ("cls");
                  } continue;
     }
     system ( "cls" );      Sleep ( 10 );
   }
}
```

O código – Versão 2

A Parte do Cliente

A parte do cliente não mudou. Essencialmente, é o mesmo código

Problemas com o Dev

- Quando for criado um projeto com o Dev++, pode ocorrer erro de ligação ("linkedição") em algumas versões desse.
- Cf. o erro, uma solução poder ser o seguinte "procedimento":
  - Vá na opção de menu "Projeto" e depois na sub-opção de menu "Opções do Projeto" .
  - Selecione a aba "Parâmetros" e em "Linker" clique no botão "Adicionar"
  - Encontre o diretório (pasta) onde foi instalado o DevCpp (talvez na 'raiz', talvez em 'arquivos de programa').
  - Encontre o subdiretório 'Lib'.
  - Encontre e selecione o arquivo libws2_32a.
  - Clique no botão "Abrir".
  - Clique no botão "OK".
  - Vá na opção de menu "Executar" e depois na sub-opção de menu "Recompilar tudo" .